Inducing a Semantically Annotated Lexicon via Deep Variational Autoencoders and EM-Based Clustering

Maurits Bleeker
University of Amsterdam
maurits.bleeker@student.uva.nl

Thijs Scheepers
University of Amsterdam
thijs.scheepers@student.uva.nl

David Zomerdijk
University of Amsterdam
david.zomerdijk@student.auc.nl

Abstract

In this work we will replicate the EM-based clustering algorithm by Rooth et al. 1999 and will extend their work with a new clustering approach based on Variational Autoencoders as well as a new method for creating a semantic representation for a cluster using word embeddings. In both our extensions we take newer deep learning concepts and apply them to try to improve the original work. Our task is to find good annotations, i.e. clusters, for subcategorization frames retrieved from a parsed corpus. These frames contain a verb-noun pair and a subcategorized frame type from the parse, e.g. "I sneezed." where "I" is the subject-noun and "sneezed" is the verb in a frame for an intransitive verb.

1 Introduction

One of the final goals of the field of natural language processing is giving machines the ability to really grasp the semantics in natural language, i.e. the ability to understand. Being able to understand the sense of a single word is an important first step. The semantics of a specific words is very much related to its context. Words can have different senses in different contexts, and thus looking at a word's context is essential for word sense disambiguation. In this work we will specifically look at the meaning of verb-noun pairs. We can infer a lot about the semantics of a verb by directly looking at its subject and objects, and inversely we can learn a lot about the semantics of a noun by looking at it's associated verb. An obvious example would be the sentence: *"Alice* $slipped_{verb}$ *through the* $crack_{noun}$*."* and the sentence *"Bob* $smoked_{verb}$ $crack_{noun}$ *and got addicted."* where we can disambiguate, and thus infer something about the semantics, of $crack_{noun}$ by directly looking at the verb.

In the first part of this work (Section 4) we will automatically induce latent semantic classes for verb-noun pairs using our own implementation[1] of the work from Rooth et al. 1999, that has become known as *Latent Semantic Clustering* (LSC). The algorithm utilizes Expectation Maximization (Dempster, Laird, and Rubin 1977) to derive a fixed set of classes for an unlabeled dataset of verb-noun pairs by training on verb-noun pairs. The goal of the algorithm is to order the lexicon of verb-noun pairs in semantic classes that are homogeneous by their semantics, i.e. that verb-noun pairs which are closely related in meaning should be clustered in the same class.

Deep neural networks have taken the field of natural language processing by storm and yielded large improvements for various language related tasks in the recent past. In section 5 we introduce a deep learning alternative to LSC using a *Variational Autoencoder* (VAE) originally conceived by Kingma and Welling 2013 . Our VAE-model will use one-hot encoding and have the goal to reproduce three separate components of our subcategorization frame: (v^w, v^p, n). We use the latent representation z from a trained VAE for clustering.

For these task we will use both a dataset of verb and noun pairs (v, n) that are extracted from a parsed corpus (British National Corpus by Burnard 2000) by using the CoreNLP by Manning et al. 2014. For every verb we extract its sub-categorization as well as the head noun of the subject NP or object NP.

To evaluate our clustering methods quantitatively we employ pseudo-disambiguation as described by Rooth et al. 1999 for both LSC as well as our VAE (Section 6). In order to apply pseudo-disambiguation to our VAE we created a a latent representation from only a noun and used the predictions from the decoder for the two required verbs. As a qualitative analysis we show some tables of the most likely nouns and verbs given a specific class.

Aside from reproducing the original LSC from Rooth et al. 1999 we also reproduced their results for lexicon induction based on the latent classes in section 7. We show qualitative results that are quite similar to the ones of the original paper, even though our dataset was constructed differently.

Lastly we combine the parameters learned from the LSC with word embeddings learned using deep learning (Mikolov et al. 2013; Pennington, Socher, and Manning 2014) to compose class embeddings (Section 8). These embeddings represent the average semantics of a specific class and could be used

1 In order to start using our implementation, first download the source code from `http://github.com/DavidZomerdijk/ULL2017`. Run the program by calling `python src/main.py`. There are additional instruction as well as documentation in the file itself.

for word sense disambiguation when working with embeddings. However we left the actual application of these embeddings and their evaluation to future work.

2 Related work

The paper by Rooth et al. 1999, which this work is based on, was back in 1999 way ahead of its time. It introduced an unsupervised data backed approach to clustering in a time when other researchers were still focusing on approaches more related to linguistic theory such as clustering based on Levin's verb classes (Levin 1993). An example was the work by Lapata and Brew 2004 which still mainly relies on the Levin's verb classes instead of learning a nice class distribution from the data.

General clustering using VAE based models has been tried by Dilokthanakul et al. 2016 and jiang et al. 2016. Both approaches use Gaussian Mixture Models (GMM) to represent the clusters within the VAE, this differs from our approach since we use a plain VAE and cluster based on the latent representation z. A logical next step for future work is to incorporate GMM into our VAE model.

To the best of our knowledge verb class disambiguation or subcategorization frame clustering has not been tried using a VAE. Hence this work will be the first trying find such a clustering method using a VAE.

3 Dataset

In order to perform our clustering methods on frames we first need to extract such frames from a parsed corpus. In the original paper these were obtained using frame induction by Carroll and Rooth 1998 on data from the British National Corpus (Described by Burnard 2000) which contains 117 million words.

For our dataset we also used the British National Corpus but extracted the frames using the CoreNLP by Manning et al. 2014. As a development set we use a set of hand-annotated pairs extracted from the Penn Treebank by Marcus et al. 1994.

Our frames are pairs of a verb v^w (including a frame categorization v^p and a noun n. So we have pairs (v, n) where $v = (v^w, v^p)$. Subcategorization for verbs describes the verb's ability to have syntactic arguments with which it co-occurs. In our case these are the head nouns from the subject NP or object NP.

Preprocessing

To keep training manageable we preprocess the data after extraction and parsing from the BNC. First, we removed all the verb-noun pairs from the dataset which contain an auxiliary an verb.

Based on intermediate results which we obtained while experimenting we discovered that axillary verbs are very likely for almost every class and therefore they do not contain much semantic information. The nature of auxiliary verbs, also so called helping verbs, is to add to another words semantics to form a coherent sentence, but is not the most important word in a sentence.

Second, we formatted all the verbs and nouns in lowercase strings to reduce input dimensionality. We discovered that some words start with an uppercase token and therefore this word will be encountered as a different string than the same word in lowercase. We also replaced all numbers in the dataset for the token *DIGIT*. An example of the latter is present in table 1.

Lastly, to further reduce the input dimensionality the verbs are stemmed using the Porter stemmer by Van Rijsbergen, Robertson, and Porter 1980 which is available in the NLTK library (Bird 2006). By stemming one reduces the number of input tokens by only using the stem of every word. This input reduction is mainly done to decrease the input size for the VEA. When the input dimensionality is smaller it becomes computational more feasible to train a VEA, due the large amount of parameters of the model. We should point out that stemming is not always done optimally. An example of this is present in table 1 which contains both verbs *write* and *wrote*.

Final processed dataset

We used $2.002.378$ frames from the parsed BNC corpus. We found 943.454 unique verb-noun pairs. And are left with a dataset of $|V|= 61.751$ unique subcategorized verbs, $|V^w|= 9.228$ unique general verbs, $|V^p|= 318$ unique subcategorizations and $|N|= 78.361$ unique nouns after preprocessing. The above techniques clearly brought down the number parameters in the model thereby making it less sparse.

We found 30.121 unique intransitive subject nouns, 34.344 unique transitive subject nouns and 34.390 unique transitive object nouns. These are later used in section 7 for lexical induction. One remark we have to make is that lines in the extracted BNC parse are structured and consecutive lines contain information about the same frame. In this way we can couple subject and object nouns from the same frame in our dataset.

In order to create class embeddings in section 8 we used GloVe embeddings (840B tokens, 2.2M vocab, 300d vectors, 5GB; by Pennington, Socher, and Manning 2014) and removed verbs and nouns without an embedding. We accounted for stemming by performing a weighted average on embeddings for the unstemmed verbs. This all left us with 8.395 embeddings for unique verbs, and 58.613 embeddings for unique nouns.

Finally from the original corpus of $2.002.378$ verb-noun pairs we separated 10.000 pairs to create an evaluation set for pseudo-disambiguation. Because lines from the BNC parse had consecutive dependence and we needed this for subject and object nouns couples we did not shuffle the dataset before extracting the evaluation data. This decision might

have slightly biased our evaluation set.

4 Latent Semantic Clustering

Here we will describe Latent Semantic Clustering (LSC), its mathematical foundation and our implementation. The goal of our LSC algorithm is to find maximize the likelihood of all the pairs in the dataset. In the process we hope to find a nice class distribution in the latent space where each class should correspond to nice semantic cluster.

The algorithm is a variant of an Expectation Maximization (EM) algorithm (Originally by Dempster, Laird, and Rubin 1977) which iteratively finds locally maximum likelihood parameters of our model. One of the nice properties of EM is that such algorithms are guaranteed to converge to a local maximum.

EM Algorithm

First we have a dataset of verb-noun pairs extracted (v, n) as described in the previous section. For every class j we have a probability distribution over unique pairs $y = (v, n)$ where $y \in \mathcal{Y}$. The set of different classes we will write as C, where $c \in C$.

The log likelihood of the data set with verb-noun pairs given the current latent θ model will converge while running the algorithm. It can be defined as:

$$\begin{aligned} L(\theta) &= \ln \prod_{(v,n)\in\mathcal{Y}} p_\theta(v,n)^{f(v,n)} \\ &= \ln \prod_{(v,n)\in\mathcal{Y}} (\sum_{c\in C} p_\theta(c_j|v,n))^{f(v,n)} \\ &= \sum_{(v,n)\in\mathcal{Y}} f(v,n) \ln \sum_{c\in C} p_\theta(c_j|v,n) \end{aligned} \tag{1}$$

Lets call x a unobserved complete data point $(c, v, n) \in \mathcal{X}$. To get the probability of a class c given a specific pair (v, n) we use the following equation:

$$\begin{aligned} p_\theta(x|y) &= p_\theta(c,v,n|v,n) \\ &= p_\theta(c|v,n) \\ &= \frac{p_\theta(c,v,n)}{p_\theta(v,n)} \\ &= \frac{p_\theta(c)p_\theta(v|c)p_\theta(n|c)}{\sum_{c'\in C} p_\theta(c')p_\theta(v|c')p_\theta(n|c')} \end{aligned} \tag{2}$$

We need the probability of a class given a pair in the M-step. During the algorithm we update the parameters $\theta = \{\theta_{vc}, \theta_{nc}, \theta_c\}$ in each iteration using the following equations. θ_{vc} is the probability distribution $p_\theta(v|c)$, θ_{nc} is $p_\theta(n|c)$ and θ_c is $p_\theta(c)$. The M-Step will result in the following equations:

$$M(\theta_{vc}) = \frac{\sum_{(v,n)\in\{v\}\times N} f(v,n)p_\theta(c|v,n)}{\sum_{(v,n)\in\mathcal{Y}} f(v,n)p_\theta(c|v,n)} \tag{3}$$

$$M(\theta_{nc}) = \frac{\sum_{(v,n)\in V\times\{n\}} f(v,n)p_\theta(c|v,n)}{\sum_{(v,n)\in\mathcal{Y}} f(v,n)p_\theta(c|v,n)} \tag{4}$$

$$M(\theta_c) = \frac{\sum_{(v,n)\in\mathcal{Y}} f(v,n)p_\theta(c|v,n)}{|\mathcal{Y}|} \tag{5}$$

In the M-Step the function $f(v, n)$ denotes the frequency of the pair in the dataset.

Now that we have defined the algorithm we can loop over the data for a certain number of steps. After each iteration we update the parameters and are able to have a better expectation for both $L(\theta)$ and $p_\theta(c|v, n)$ and thus we will gradually converge to a locally optimal solution.

Implementation

Our implementation[2] is written in python with the use of the NumPy[3] linear algebra library. The result of using just basic libraries makes sure our code is general and understandable.

As default we extract $|C|= 50$ classes from the dataset, which results in three matrices, one $|V|\times|C|$ matrix for the distribution $p_\theta(c|v)$ for θ_{vc}, one $|N|\times|C|$ matrix for the distribution $p_\theta(c|n)$ for θ_{nc} and one $|C|$ vector for θ_c. We randomly initialize this matrix, and divide every column value trough the sum of the whole column to get a random probability distribution, i.e. each class will sum to 1.

Our in memory loaded dataset [4] already prepared the data in such a way our implementation of LSC could execute with efficiency. Additionally, our implementation is optimized for speed using dynamic programming principles, and fast NumPy matrix calculations. We used lists to access matrices at specific indices which is also implemented within the NumPy library making our implementation a lot faster than a naive one. Inducing 50 classes from the data for 100 iterations took 820 seconds runtime on a Apple MacBook Pro A1502 with a Intel Core i5-5257U.

Lastly our implementation makes it easy to store intermediate results and model state, and plug evaluation measures into the training procedure.

5 Variational Autoencoder

In this section we first describe the variational autoencoder as proposed by Kingma and Welling 2013 after which we describe how we modified its more traditional architecture (with continuous input representation instead of one-hot vectors) to allow it to effectively reconstruct verb-noun pairs $\hat{x} \approx (v^w, v^p, n)$ from an input $x = (v^w, v^p, n)$. The constructed latent representation z could be seen as a latent representation for our verb-noun pairs x, which could be used for clustering.

2 Each component of the implementations is split up in different class objects, the actual model described in this section is implemented in the class `LSCVerbClasses`.

3 The linear algebra python library, NumPy `http://www.numpy.org`

4 The implemenation of the dataset can be found in the class `Dataset`.

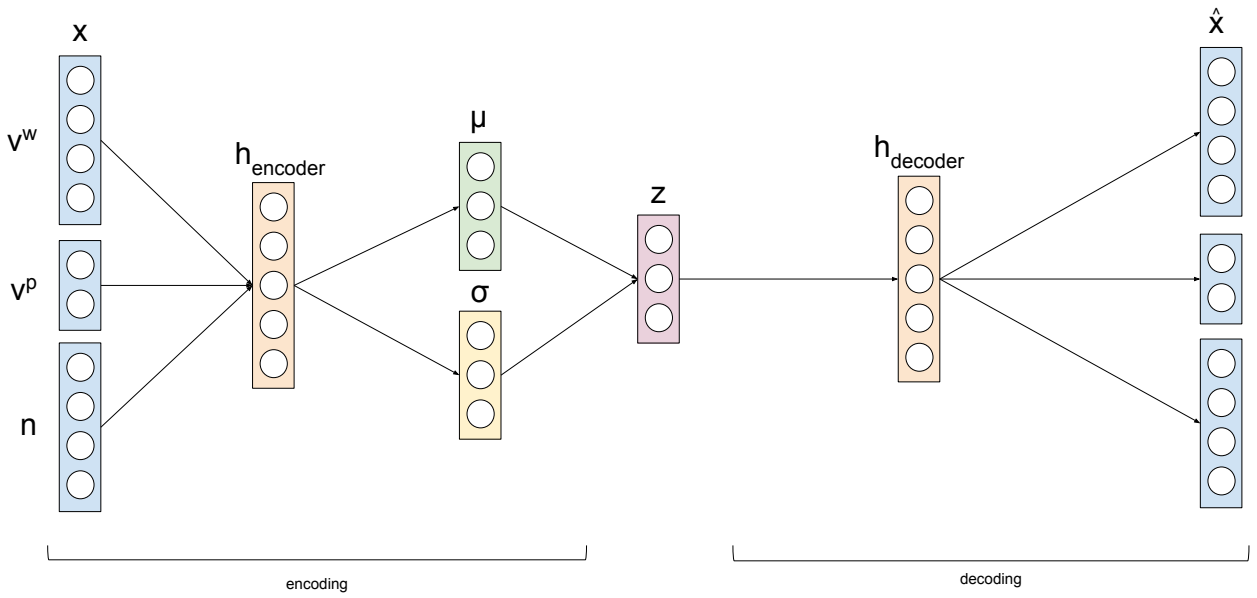


FIGURE 1: The architecture of the VAE. The figure shows (v^w, v^p, n) as three input vectors which are all three one hot vectors, encoded in to a single embedding at layer $h_{encoder}$, which then enters the layers for μ and σ to encode in the latent representation z, which is than decoded into an embedding again at $h_{decoder}$ which we use to predict $\hat{x} \approx (v^w, v^p, n)$ using a softmax function for each of the three components separately.

Background

The general aim of an autoencoder is to encode the input to a lower dimensional latent representation. To learn this representation the autoencoder also has a decoder which aims to reconstruct the input from the latent representation. Consequently, the latent representation can be learned in an unsupervised manner by training it to minimize a reconstruction error. However, the problem of this approach is that it generalizes poorly. This is because the autoencoder can learn one-to-one mappings to minimize the reconstruction loss, which is not desired since it will consequently perform poorly on unseen data.

To solve this problem Kingma and Welling 2013 proposed the VAE that constrains the encoding of the latent vector to have values that follow a Gaussian distribution. In this way the inputs that are similar will be forced to be closer together in the latent space. To enforce this constraint the Kulback-Leibler (KL) divergence is added to loss function as can be viewed in equation 6. The first part of the loss is the expected negative log likelihood, also called the reconstruction loss since it penalizes the objective when the reconstructed output $\hat{x}$ is different from the input x. The second term is the KL divergence which measures how close a distribution q is to p. Which basically means how much information is lost when one uses q to represent p. In our case we are interested in how close the encoders distribution $q_\theta(z|x_i)$ is to $p(z)$. In the case of the VAE $p(z)$ is chosen to be a Gaussian with mean zero and a variance of 1. Thus $p(z) = \mathcal{N}(0, 1)$. This means that the KL-divergence enforces that the the latent representation of the VAE follows a standard Normal distribution. To guide this process the latent representation z is sampled from a vector μ that represents the mean of the Gaussian and a vector σ that represents the standard deviation of the Gaussian. Thus it's not the latent representation that is learned by the encoder, but the parameters of a Gaussian. Since the standard variational autoencoder is created to process image data we had to make some adjustments to have it work properly for our application.

Implementation

In figure 1 the architecture of our model is depicted. To represent the verb noun pair (v, n) we chose to represent the verbs and the nouns with a one-hot encoding. In the preprocessing step we removed the relation that is attached to the verb and also created a one-hot encoding for this. We concatenated these three vectors to represent the input. To be able to reconstruct the one hot encodings for (v^w, v^p, n) properly we had to adjust the reconstruction loss. Thus instead of using the loss function as described in equation 6 we used three sparse softmax entropy functions for the range that represents the three separate components of our sub categorization frame: (v^w, v^p, n) as is depicted in equation 7.

$$l_i(\theta, \phi) = -E_{z\sim q_\theta(z|x_i)}[\log p_\phi(x_i|z)] + KL(q_\theta(z|x_i)||p(z)) \tag{6}$$

Some other details of our implementation:
Both the size of our hidden encoder layer and hidden decoder layer is 600. Our hidden units used the ReLU activation function (Nair and G. E. Hinton 2010). We used a latent vector z of size 50 because 50 classes yielded the best results for the EM algorithm. Consequently, vector μ and σ are also of size 50. We used a batch size of 1024. We regu-

TABLE 1: This table shows the probabilities of c_7 after 100 iterations of training. Dots in the table show if the noun-verb pair was present in the dataset. These are the 20 most likely nouns and verbs. We can clearly see this class represents all things related to *writing and publishing*. Another remark we could make here is that we see our *DIGIT* token from preprocessing in the list of most likely nouns.

Class 7	θ_{nc}	0.024	0.020	0.016	0.016	0.015	0.012	0.011	0.009	0.009	0.008	0.008	0.008	0.008	0.008	0.007	0.007	0.007	0.006	0.006	0.006
θ_{vc}	$\theta_c = 0.017$	page	chapter	table	figure	book	report	books	p.	list	letter	section	DIGIT	figures	paper	article	letters	fig.	pages	reports	results
0.134	see.o_dobj	•	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•	•
0.075	includ.so_dobj		•	•	•	•	•	•		•	•	•	•	•	•	•	•		•	•	•
0.071	show.so_nsubj		•	•	•	•	•	•		•	•	•	•	•	•	•		•	•	•	•
0.031	includ.soooo_dobj							•					•				•				
0.023	wrote.so_dobj		•			•	•	•		•	•		•		•	•	•			•	
0.016	publish.so_dobj					•	•	•		•	•		•	•	•	•	•			•	•
0.015	write.so_dobj	•	•		•	•	•	•		•	•		•		•	•	•		•	•	•
0.014	produc.so_dobj			•	•	•	•	•		•	•		•	•	•	•	•		•	•	•
0.013	contain.so_dobj	•	•	•	•	•	•	•		•	•	•	•	•			•		•	•	•
0.012	read.so_dobj	•	•		•	•	•	•		•	•	•	•	•	•	•	•		•	•	•
0.011	see.so_dobj	•	•	•	•	•	•	•	•	•	•	•	•	•		•	•	•		•	•
0.010	see.oo_dobj	•	•	•	•	•			•			•	•					•	•		
0.010	give.so_nsubj	•	•	•	•	•	•	•		•	•	•	•	•	•	•	•		•	•	•
0.010	appear.s_nsubj	•			•	•	•	•		•	•		•	•	•	•	•			•	•
0.009	show.s_nsubj		•	•	•		•					•	•	•	•		•		•		•
0.009	read.o_dobj	•	•	•		•	•	•			•	•	•		•	•	•		•	•	
0.008	illustr.so_nsubj	•	•	•	•	•		•			•			•	•	•		•		•	•
0.007	contain.so_nsubj	•	•	•	•	•	•	•		•	•	•	•	•	•	•	•		•	•	•
0.007	write.o_dobj	•				•	•	•		•	•	•	•		•		•			•	
0.007	show.soo_nsubj			•	•	•	•						•	•				•			•

larized our loss function using 0.001 weighted L2-regularization. Lastly, our implementation used the Tensorflow library (Abadi et al. 2016) and is based on the implementation of the standard VAE by Van Amersfoort 2016.

$$\begin{aligned} l_i(\theta, \phi) = & softmax(\hat{x}[v^w_{start}, v^w_{end}]) \\ & + softmax(\hat{x}[v^p_{start}, v^p_{end}]) \\ & + softmax(\hat{x}[n_{start}, n_{end}]) \\ & + KL(q_\theta(z|x_i)||p(z)) \end{aligned} \tag{7}$$

Training the model for 100 epochs (total steps 91270 with a batch size of 1024) and executing test runs every (100 steps) took about 10 hours on a nVidia GeForce GTX 980Ti GPU.

6 Evaluation

Now that we have defined both the LSC model as well as the VAE model we can run experiments using our dataset. In this section we will both use a qualitative and quantitative approach to evaluate the models. Both models will be evaluated quantitatively using pseudo-disambiguation as described by Rooth et al. 1999. We evaluate the LSC model qualitatively in a similar manner as the original paper. For the VAE we also include reconstruction accuracy which tells us a lot about the information available in the latent space z.

6.1 Pseudo-disambiguation

In this section we first describe how we applied it to the clusters from LSC after which we describe how we applied it to validate the quality of our VAE model.

From the original corpus of $2.002.378$ verb-noun pairs we separated 10.000 pairs to create an evaluation set. The remaining pairs are used as a training set and contains $1.147.763$ unique pairs. From the 10.000 separated pairs all verbs and nouns are removed that occur less than 30 times or more than 10.000 times in the train set. Doubles are removed as well. This results in an evaluation set of 3112 unique verb-noun pairs. These tuples are appended with a randomly sampled verb v' that does adhere to the following properties. v' should occur more than 30 times and less than 3.000 times in the train set, as well as the tuple (v', n) should not be present in the entire dataset.

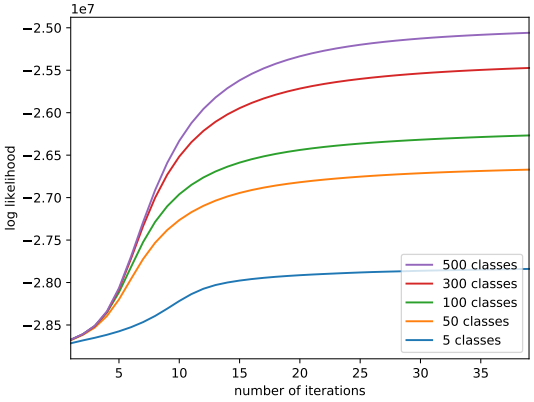
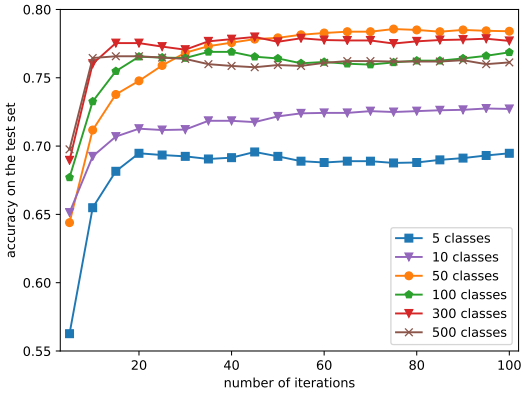


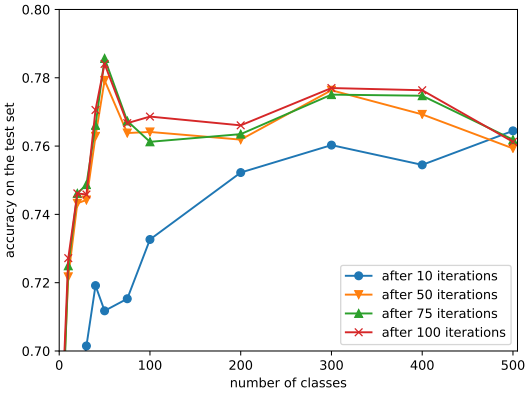
FIGURE 2: Likelihood convergence for the LSC models. For all models the likelihood converges steeply between 5 and 10 iterations from around -3 ten-millionth to various local optima, e.g. -2.5 ten-millionth for 500 classes.

For these triples (v, n, v') we check whether $p(n|v) \geq p(n|v')$. The number of triples for which this is the case divided by the total number in the evaluation set is defined as the accuracy.

For LSC we calculated $p(n|v)$ and $p(n|v')$ using the parameters in our model as described in equation 8. For the VAE we find $p(n|v)$ and $p(n|v')$ by inserting



(A) Accuracy for different numbers of classes at each iteration: For all numbers of classes the accuracy converges after more iterations.



(B) Accuracy in terms of the number of classes. The accuracy continues to increase until 100 classes after which the accuracy slowly decreases.

FIGURE 3: Pseudo-disambiguation accuracy $(p(n|v) > p(n|v'))$ for the LSC models.

only the verb vector into the VAE (in this case the concatenation of v^w and v^p), hereby keeping all values of the noun vector n at zero. Then we evaluate the softmax value for the noun for both verb v and the random verb v' to get $p(n|v)$ and $p(n|v')$. Thus, for the pseudo-disambiguation for the VAE we employ softmax probabilities. The mechanism is similar and we therefore think we are still able to compare the pseudo-disambiguation accuracy between LSC and VAE. Additionally we also perform inverse pseudo disambiguation on the VAE where we compare $p(v|n)$ and $p(v'|n)$ using the same dataset.

$$p(n|v) = \frac{\sum_c p_\theta(c)p_\theta(n|c)p_\theta(v|c)}{\sum_n \sum_c p_\theta(c)p_\theta(n|c)p_\theta(v|c)} \tag{8}$$

The accuracy shows whether existing verb noun pairs seem more probable to our model than random verb noun pairs, hereby validating whether our model finds relevant classes that generalize over unseen verb noun pairs.

6.2 Evaluating LSC

Experimental setup

We train our EM algorithm for 150 iterations for various class configurations: $|C| = \{5, 10, 20, 30, 40, 50, 75, 100, 200, 300, 400, 500\}$. Figure 2 shows the progression for various class numbers, and we can clearly see that more classes yield a higher log-likelihood, which is what you expect with more parameters.

Evaluation

Just looking at the likelihood progression is obviously not the right approach to evaluate the model. In the next section we will calculate accuracy using pseudo-disambiguation by which we should be able to quantify the quality of the classes.

Table 1 shows the seventh induced semantic class from a model with 50 classes. The rows list the 20 most probable verbs, and the columns list the 20 most probable nouns. The verb-noun pairs that were present in the dataset are indicated by a dot. As one can observe in table 1 the seventh induced class seems to be a valid cluster containing words that have a relation with written works. We see verbs as wrote, write, read and illustrate and nouns as page, chapter, report and books. This example shows that our model was able to find proper semantic verb-noun classes.

In figure 2 one observe how the likelihood increases rapidly after which it converges in the consecutive 20 iterations. As one would expect we observe that a higher number of clusters yield a higher log likelihood.

We trained 12 different models using different cluster sizes for 100 iterations. Figure 3a shows how the accuracy improves after training for all number of classes and converge within 20 EM iterations to different levels of accuracy.

In figure 3b the accuracy is depicted for different number of classes after 10, 50, 75 and 100 iterations. The accuracy increases with the number of classes up until 50 classes which yields an accuracy of 79%[5]. Thereafter the accuracy slowly decreases which is probably caused by overfitting as also mentioned by Rooth et al. 1999. Also it is interesting that we have the best result after only 75 iterations instead of a 100, which might also indicate some overfitting.

Because we found 50 classes to yield the best result we will compare everything to this LSC model for the next parts of this work. The parameters of the VAE are chosen based on the 50 classes, we use the

5 We also ran experiments with a dataset where we did not stem and lowercase and got an accuracy of 81% for 100 classes. This can be explained due to the fact we removed distinctive characteristics from the dataset. However we needed to lower the input dimensions for the VAE to make it trainable and in order to compare it to the LSC we also used this final dataset for the LSC models.

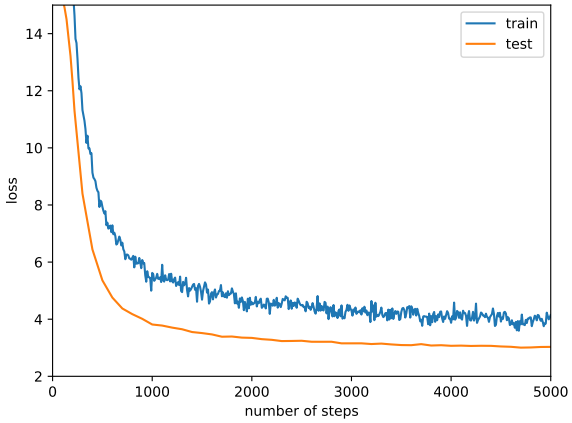


FIGURE 4: This figure shows the loss during training on the training batches, and the complete test set. Interestingly the loss on the test set is lower, this can be explained by the way we constructed the test set, as we had to do this the same way to allow for comparison with the LSC model.

model for lexicon induction in section 7 as well as for the class embeddings from section 8.

6.3 Evaluating VAE

Experimental setup

To evaluate the quality of the latent representations from our VAE we will look at the reconstruction accuracy, pseudo-disambiguation accuracy and by analyzing clusters that emerge from clustering the latent space of the VAE using k-means clustering. We trained the VAE for 100 epochs after which the loss on both the train and the test set have converged. We chose to do 100 epochs to let the model take a look at the data a similar amount to the LSC model.

The loss of the VAE is depicted in figure 5a for the first 5000 of the 91270 steps which shows that the loss has smoothly converged. Every step on the x axis represents another training step on a batch of 1024 data points. Interestingly the loss on the test set is lower, this can be explained by the way we constructed the test set, as we had to do this the same way to allow for comparison with Rooth et al. 1999 as described in section 3. From this combined with converging reconstruction accuracy we can conclude that the way we adjusted the VAE results in a stable method to reconstruct verb noun representations. In the next sections we will evaluate the quality of the created latent representations from the VAE.

Reconstruction accuracy

Since the VAE is trained to create proper reconstructions from the data let us first look at how well the model is able to reconstruct unseen verb-noun pairs. In Figure 5a the reconstruction error for verbs, nouns and sub-categorization frame is depicted and shows that the VAE rather quickly is able to reconstruct the verb-noun pair. Again this figure focuses on the first for the first 5000 of the 91270 steps. We find the

TABLE 2: These are the top 10 most likely nouns given cluster 48 $p(n|c_{48})$ from k-means over the latent space z of the VAE.

nouns from class: 48			
support	0.035	home	0.031
book	0.033	boy	0.030
father	0.033	other	0.030
changes	0.033	world	0.028
experience	0.32	business	0.028

TABLE 3: These are the top 10 most likely verbs given cluster 48 $p(v|c_{48})$ from k-means over the latent space z of the VAE.

verbs from class: 48			
view	0.041	drove	0.032
overlook	0.033	assum	0.032
emphasis	0.033	releas	0.030
repli	0.033	expect	0.030
sign	0.33	succeed	0.030

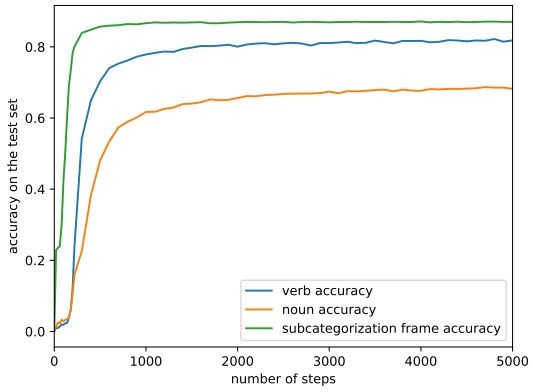
following accuracy after convergence: ($\hat{v}^{w}_{accuracy} = 0.83, \hat{n}_{accuracy} = 0.71, \hat{v}^{p}_{accuracy} = 0.87$). The subcategorization accuracy is highest which is caused by the lower dimensionality of the output vector, therefore the chance of predicting correct is higher. Consequently, the noun accuracy is lowest since it has the largest output vector. From this we can conclude that the VAE is able to reconstruct the input and does not overfit. We determined the accuracy by comparing the input vectors to the reconstructed $\hat{x}$ with the highest softmax value.

Pseudo-disambiguation accuracy

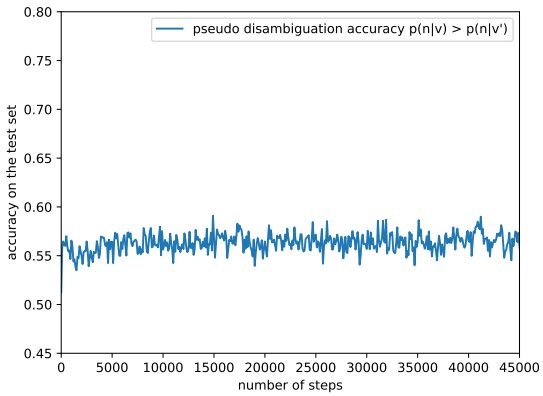
To compare the VAE with LSC we performed pseudo-disambiguation. Figure 5b depicts the pseudo-disambiguation progression for the first 45000 of the 91270 steps over the training data. This accuracy shows that it gradually increases but never achieves a performance close to that of EM. It's maximum performance is 0.59 which is a lot lower than the 0.79 of EM.

To see whether the low pseudo-disambiguation accuracy is caused by the high dimensionality of the noun vector n we also performed inverted pseudo disambiguation where looked at $(p(v|n) > p(v'|n))$. For this we fed the VAE only the noun vectors after which we compared the softmax value for v and v'. Figure 5c shows that this achieves higher accuracies which indicates that the size of the output influences the pseudo-disambiguation accuracy.

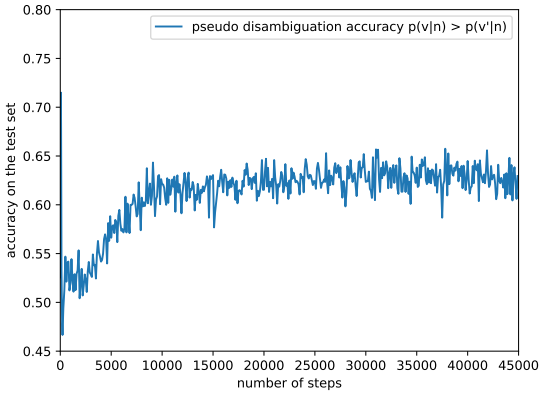
Figure 5c depicts the first 250 steps from figure 5c. In the first 100 steps the pseudo-disambiguation peeks to 0.72 after which it drops back below 0.50 from which it converges back to 0.66. The behavior during the first 200 steps is remarkable, especially since at this point the VAE hasn't observed all the data yet. This strange artifact could be caused by how we create our test set but have no proof to back this explanation. Or maybe a really quickly apparent over-fitting effect. Although, it seems that the EM outperforms the VAE, it might just be that our



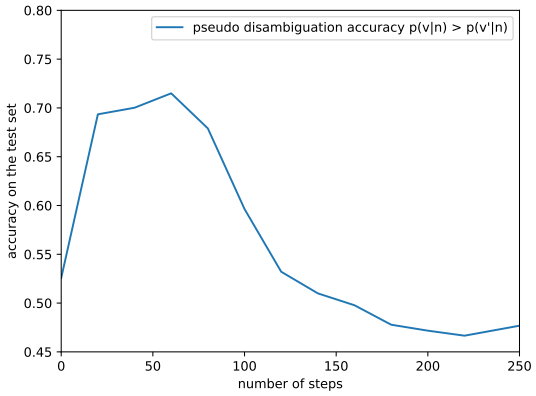
(A) This figure shows the reconstruction accuracy on the test set for the VAE. We measure reconstruction accuracy for the three input and output components individually.



(B) 45000 steps of the Pseudo-disambiguation accuracy $(p(n|v) > p(n|v'))$ for the VAE. Peaked at 0.591.



(C) 45000 steps of the inverted pseudo-disambiguation accuracy $(p(v|n) > p(v'|n))$ for the VAE. Peaked at 0.657.



(D) First 250 steps of the inverted pseudo-disambiguation accuracy $(p(v|n) > p(v'|n))$ for the variational auto encoder. It is interesting that we see this spike early in training. Peaked at 0.715.

FIGURE 5: Accuracy for the VAE models.

way of applying pseudo-disambiguation to the VAE is not a valid way to actually compare both methods.

Clustering based on the latent representation

Lastly, we clustered the latent representations z of the VAE into 50 clusters using the k-means algorithm from the Scikit-Learn library (Pedregosa et al. 2011). To assess the quality of these clusters we look at the most probable verbs in the clusters, $p(v|c)$ and by looking at the most probable nouns in the clusters $p(n|c)$. To find these probabilities we calculated $f(v,c)/(v)$ and $f(n,c)/f(n)$ for all verbs and nouns in all clusters.

Table 2 shows the most probable nouns for cluster 48 and 3 shows the most probable verbs for cluster 48. The nouns seem somewhat related since *support*, *father*, *home*, and *boy* are all related to family situations. However, they are far from as clear as the ones from EM. The most probable verb classes don't seem to have any relation with each other. Thus, after inspecting the incoherent verbs and nouns in the k-means clusters we have to conclude that k-means over latent vectors of the VAE isn't able to extract proper semantic clusters for the verb-noun pairs.

In section 9 we make several suggestions on how we could improve the VAE and its put clustering ability directly into the model, so the need for applying k-means is left out and we can train on clusters directly end-to-end. This could be accomplished by using a mixture of Gaussians.

7 Lexicon Induction

In addition to replicating the LSC from Rooth et al. 1999 and evaluation, we try to replicate their results with respect to lexicon induction in this section. When we have $p(n|c)$ from the LSC model with fixed parameters we further optimize the classes to find a distribution over nouns that are in (v,n) pairs where the verb is intransitive.

We start by defining the log-likelihood:

$$L(\theta) = \sum_{n \in N} f(n) \ln \sum_{c' \in C} p(c) p_{LSC}(n|c') \qquad (9)$$

After which we define the probability distribution of $p(n)$:

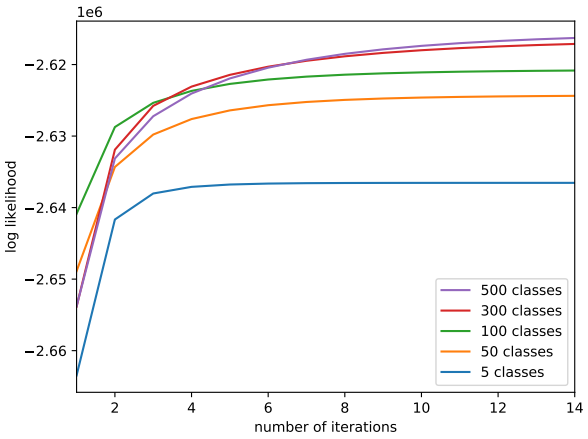


FIGURE 6: Likelihood convergence for lexicon induction: After 3 iterations the likelihood converges around -3.8 million for intransitive verbs. (*y*-axis depicts the likelihood)

$$p(n) = \sum_{c \in C} p(c, n) = \sum_{c \in C} p(c) p_{LSC}(n|c) \tag{10}$$

And to continue we express it in terms of $p(c|n)$:

$$p_\theta(c|n) = \frac{p(c) p_{LSC}(n|c)}{\sum_{c' \in C} p(c') p_{LSC}(n|c')} \tag{11}$$

In the M-step we will apply as according to equation 12. In this M-step the frequency $f(n)$ is the amount of times that noun has been seen in combination with a selected intransitive verb.

$$M(\theta_c) = \frac{\sum_{n \in N} f(n) p_\theta(c|n)}{\sum_{n \in N} f(n)} \tag{12}$$

In our implementation[6] we initialized the model randomly, but with a LSC which had been trained for 100 iterations. In the dataset we found $|N|=45.421$ unique subjects for intransitive verbs and we used these, along with their frequencies i.c.m. with the same verbs. Because this model required less parameters and less data it trained a lot faster.

Figure 6 shows the likelihood progression for the model that finds classes for nouns paired to intransitive verbs and this is increasing, again we see that additional classes lets this model increase higher but that is expected due to the added parameters.

Qualitative evaluation of Lexicon Induction

In table 4 and 5 an experimental evaluation for the lexicon induction has been given. Experiments used a model with 50 classes trained for 100 iterations.

Given a intransitive verb v (the stemmed word for *increase* in this case) we computed the most likely class given the verb using the model from section 4.

6 Our implementation of this model is written in the class `SubjectIntransitiveVerbClasses`, which accepts an instance of `LSCVerbClasses` for use of p_{LSC}.

TABLE 4: Most likely nouns given the intransitive verb *increase*, for the the class where increase has the highest probability $P(c|v)$. We see this class representing the *negative sense* of the verb increase.

increase	class: 48	$p(c_{48}\|v) = 0.80$	
noise	5.29	pay	0.28
drinking	1.42	houses	0.27
roar	0.50	pains	0.24
complaints	0.34	strains	0.23
aids	0.33	prices	0.22

TABLE 5: Table for the second most likely class for the verb *increase*, we see this class representing the *numeric sense* or *business sense* of the verb increase.

increase	class: 1	$p(c_1\|v) = 0.044$	
prices	291.90	sales	90.12
earnings	164.69	rate	86.05
profits	137.39	rates	84.17
shares	136.50	production	56.19
number	132.82	profit	55.59

This is done by using $p(c|v, n)$ and integrate out the nouns (eq. 13).

$$p(c|v) = \sum_n p(c|v, n) \tag{13}$$

In the tables we use the estimated frequencies of $f(n)p_\theta(c|n)$, using the $p_\theta(c|n)$ from our new lexicon induction model.

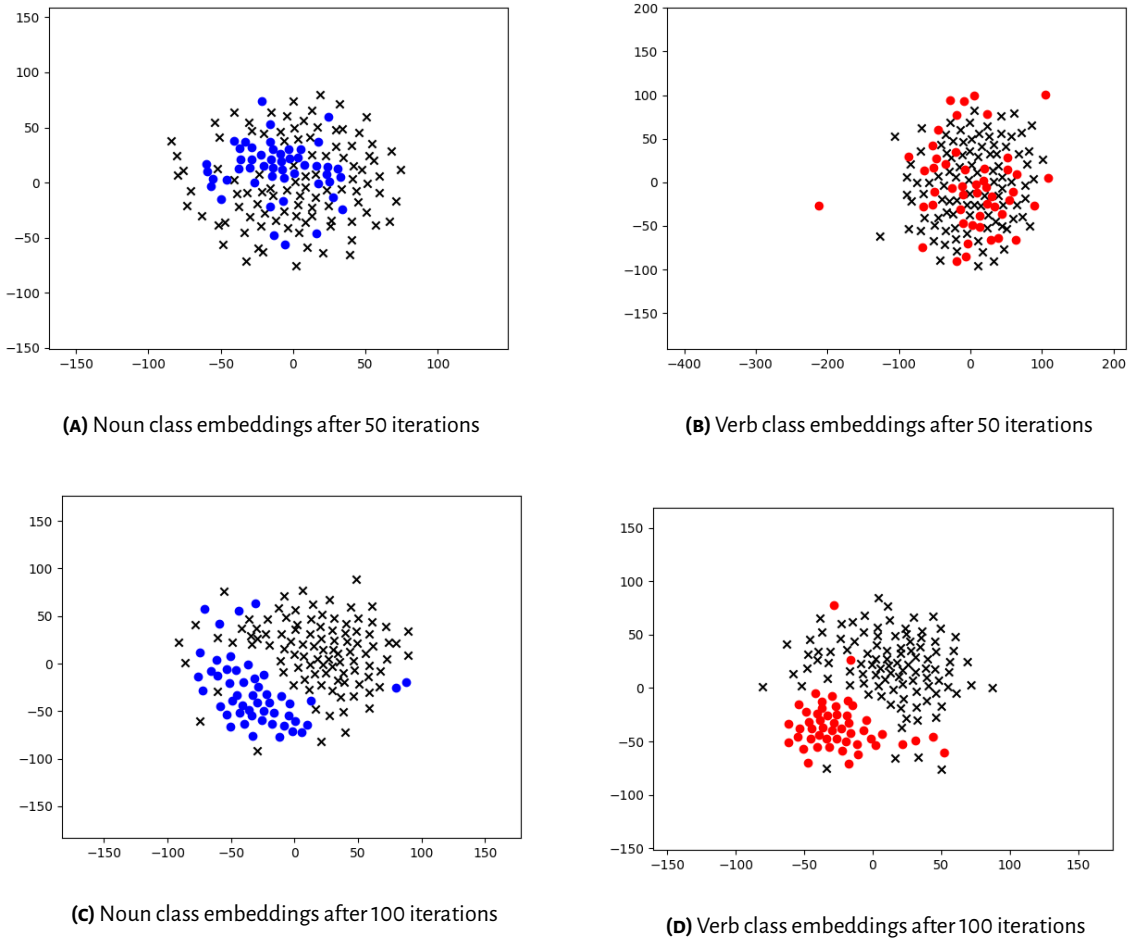
When looking at table 4 and 5 we can see that the model properly does lexicon induction. Class 48 is the most likely class given the verb *increase*. Even though the verb *increase* is the most likely for the class 48, not all the nouns with the highest frequencies $f(n)p_\theta(c|n)$ that belong to that class (see table 4) are very common to have semantic correspondence with the verb textitincrease (for example housed).

Although class 48 is more likely given the verb *increase*, we can see that especially table 5 (class 1) shows nouns that are likely to have something in common with the verb increase. For example, (increase, prices) and (increase, production), etc. Based on this table it can be concluded that our model does proper Lexicon Induction. Only it differs per class and the belonging nouns how well the results are.

When looking at both table 4 and 5 the semantic difference between two different classes become clear. Table 5 clearly shows nouns that have something in common with business and a numeric sense of increase while table 4 contains nouns denoting some negative in-numeric sense of increase.

8 Class embeddings

In this work we combine modern Deep Learning approaches with the work from Rooth et al. 1999. Word-embeddings play a huge role in natural language processing ever since the work by Mikolov et al. 2013 made them popular and described their distributional semantic properties. One way we can use the parameters from our LSC model is to combine



(A) Noun class embeddings after 50 iterations

(B) Verb class embeddings after 50 iterations

(C) Noun class embeddings after 100 iterations

(D) Verb class embeddings after 100 iterations

FIGURE 7: 2 dimensional t-SNE on the class embeddings, we see black crosses marking background embeddings, blue dots for noun classes and red dots for verb classes.

these with pre-trained word embeddings to create class-embeddings. In the ideal case, the space of all word embeddings represents all possible semantics. And thus we can create a nice semantic average for a class by which we can deduce its meaning.

We loaded word-embeddings from the pre-trained embedding set GloVe (Pennington, Socher, and Manning 2014), as described in section 3. A important remark here is that we use the same embedding for a verb within a different subcategorization frame.

Now that we have θ_{vc} for the probability distribution $p_\theta(v|c)$ and θ_{nc} for $p_\theta(n|c)$ we can use these as weights for a weighted average class embedding. The resulted embeddings can represent the average meaning of a disambiguation class, and thus can for example be used to disambiguate between word senses.

Evaluating the class embeddings

For all 50 classes we create embeddings at different parts in the training process. Earlier in section 6 we saw that the LSC model starts to overfit after 75 iterations and it looses accuracy on the pseudo-disambiguation task.

We evaluate our found class embeddings by looking at t-SNE projections (Maaten and G. Hinton 2008) that are shown in Figure 7. These t-SNE projections are stochastic in nature and are projected into two dimensions. We use 250 background embeddings to show how the class embeddings relate to the embeddings of normal words.

We can see in figures 7b and 7a that our cluster embeddings are nicely spread out in between the background embeddings, so they each have a nice specific semantic representation. However we see that when the model starts to overfit at 100 iterations in figure 7d and 7c the class embeddings start to be structured different from the background embeddings and t-SNE is able to cluster them. Usually when you apply t-SNE you would like to see clustering taking place, however in this case we want exactly the opposite since we want the cluster embeddings occupy similar space as related words that describe the clusters semantics.

9 Future work

In this section we will discuss various directions for future work.

Cluster evaluation

Pseudo-disambiguation is just one way to evaluate latent clustering models The original work by Rooth et al. 1999 also described smoothing power which we did not implement. Additionally there are more methods to evaluate clusters. For example we could evaluate the found clusters using internal metrics to see if the clusters are compact and well-separated as stated by Halkidi, Batistakis, and Vazirgiannis 2001.

Transitive verb, subject-object pairs

The original work by Rooth et al. 1999 also builds a lexicon induction model for subject-object pairs for transitive verbs. We implemented[7] this part of the paper as well, however we believe we made a slight mistake somewhere since our likelihood is not decreasing. We would like to mark this specific implementation as future work.

Experiments with different network structures and latent representation size

Due the limited amount of time for this research project, only some small experiments have been done with the hyperparameters of our VEA en LSC models. For future research different network architectures and dimensionalities of the latent presentation could be tested. Perhaps with a more sophisticated network architecture or latent space representation, it will be possible to learn more expressive latent classes.

Extend the VAE to include the entire frame

Our experiments focuses on a VAE that produces a latent representation from a fixed three slot input. One of the big advantages to deep learning models, is that they are very flexible in the way architecture is constructed. We could extend our VAE to accept the entire frame, including all object nouns and at which place they are. We could do this similar to the recursive neural networks (not recurrent) from Socher et al. 2013.

Gaussian Mixture Models for the VAE

To allow the VAE to have a more discrete notion of clusters we could include Gaussian Mixture Models into the VAE similar to the work of Dilokthanakul et al. 2016. Although the results were mixed, introducing a GMM within the VAE benefits clustering.

Using class embeddings to transform word embeddings towards a word sense embedding

First of all, the applicability of class embeddings still needs further evaluation. Perhaps looking at the closest words to a class embedding in the space which we think could be close to a word describing the class the best. But when class embeddings are seen as useful they could be used to pull word embeddings more towards their specific sense in a certain context. This can be helpful for models using embeddings such as NMT models for example.

7 The transitive verb, subject-object pairs lexicon induction is implemented in the class `SubjectObjectTransitiveVerbClasses`.

10 Conclusion

We successfully reproduced large parts of the work of Rooth et al. 1999 including both the LSC alrogithm as the Lexical Induction. We found similar patterns for pseudo-disambiguation accuracy and eventually found that 50 classes was the best LSC model for our dataset (0.79 pseudo-disambiguation accuracy).

We implemented a VAE model which obtained a high reconstruction accuracy ($\hat{v}^w_{accuracy} = 0.83, \hat{n}_{accuracy} = 0.71, \hat{v}^p_{accuracy} = 0.87$) for the test set which shows its generalizability. However when performing pseudo-disambiguation on the VAE model we see that the model does not outperform LSC (0.59). For both models we find nice classes which can be described qualitatively as fitting specific semantics.

We introduced class embeddings based on the parameters learned with the LSC model, the application of these embeddings are left for future research.

References

Abadi, Martín et al. (2016). "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467*.

Bird, Steven (2006). "NLTK: the natural language toolkit". In: *Proceedings of the COLING/ACL on Interactive presentation sessions*. Association for Computational Linguistics, pp. 69–72.

Burnard, Lou (2000). *Reference guide for the british national corpus (world edition)*.

Carroll, Glenn and Mats Rooth (1998). "Valence induction with a head-lexicalized PCFG". In: *arXiv preprint cmp-lg/9805001*.

Dempster, Arthur P, Nan M Laird, and Donald B Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38.

Dilokthanakul, Nat et al. (2016). "Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders". In: *arXiv preprint arXiv:1611.02648*.

Halkidi, Maria, Yannis Batistakis, and Michalis Vazirgiannis (2001). "On clustering validation techniques". In: *Journal of intelligent information systems* 17.2, pp. 107–145.

jiang, Zhuxi et al. (2016). "Variational Deep Embedding: A Generative Approach to Clustering". In: *CoRR* abs/1611.05148.

Kingma, Diederik P and Max Welling (2013). "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114*.

Lapata, Mirella and Chris Brew (2004). "Verb class disambiguation using informative priors". In: *Computational Linguistics* 30.1, pp. 45–73.

Levin, Beth (1993). *English verb classes and alternations: A preliminary investigation*. University of Chicago press.

Maaten, Laurens van der and Geoffrey Hinton (2008). "Visualizing data using t-SNE". In: *Journal of Machine Learning Research* 9.Nov, pp. 2579–2605.

Manning, Christopher D. et al. (2014). "The Stanford CoreNLP Natural Language Processing Toolkit". In: *Association for Computational Linguistics (ACL) System Demonstrations*, pp. 55–60.

Marcus, Mitchell et al. (1994). "The Penn Treebank: Annotating Predicate Argument Structure". In: *Proceedings of the Workshop on Human Language Technology*. HLT '94. Plainsboro, NJ: Association for Computational Linguistics, pp. 114–119. DOI: 10.3115/1075812.1075835.

Mikolov, Tomas et al. (2013). "Efficient estimation of word representations in vector space". In:

Nair, Vinod and Geoffrey E Hinton (2010). "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.

Pedregosa, Fabian et al. (2011). "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research* 12.Oct, pp. 2825–2830.

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning (2014). "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*.

Rooth, Mats et al. (1999). "Inducing a semantically annotated lexicon via EM-based clustering". In: *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*. Association for Computational Linguistics, pp. 104–111.

Socher, Richard et al. (2013). "Parsing with Compositional Vector Grammars." In: *ACL (1)*, pp. 455–465.

Van Amersfoort, J. (2016). *VAE-TensorFlow*. https://github.com/y0ast/VAE-TensorFlow.

Van Rijsbergen, Cornelis J, Stephen Edward Robertson, and Martin F Porter (1980). *New models in probabilistic information retrieval*. British Library Research and Development Department.